

## Background

First, in order to validate a client certificate the server (or load balancer) will need to have the root certificate that signed the client certificate installed. So a CA issues a root certificate. The public key of that root certificate signs the client certificate. Typically, the root CA does not sign server or client certificates directly. The root CA is only ever used to create one or more intermediate CAs, which are trusted by the root CA to sign certificates on their behalf. When the client authenticates the server can verify that the client certificate really was signed by the CA. And thereby trust it.

It is common that load balancers verify the client certificate. You can usually figure out what load balancer it is by looking at response headers and cookies.

For example F5 load balancers usually tests a cookie that looks something like this:

```
TS*****
TS01*****
```

The load balancers usually verifies the client certificate, retrieve some information from the certificate and then pass it along to the application. The information sent from the load balancer to the application server is usually sent in a specific header. Here is a list of common headers that carries this information. Copied from here: <https://github.com/wallarm/cert-headers/blob/master/headers.list>.

```
#F5
SSLClientCertStatus
SSLClientCertSN
SSLClientCertb64
X-Client-Cert
X-Client-Certificate
X-Client-Crt
#HAProxy
X-SSL
X-SSL-Client-Cert
X-SSL-Client-Verify
X-SSL-Client-SHA1
X-SSL-Client-DN
X-SSL-Client-CN
X-SSL-Issuer
X-SSL-Client-Not-Before
X-SSL-Client-Not-After
X-SSL-Client-Serial
X-SSL-Client-Version
X-Valid-Certificate
#Nginx
X-SSL-CERT
X-SSL-VERIFIED
X-SSL-CLIENT-DN
X-SSL-ISSUER-DN
```

## PKCS # 12

PKCS (Public-Key Cryptography Standard) # 12 is an archive format. It is used to store two things. A public X509 certificate and a private key, or to bundle all members of a chain of trust. The format is binary and can be read using openssl. It would be nice if the data would be stored in a semi-normal format such as JSON or XML, but it isn't.

The format is described in RFC 7292: <https://tools.ietf.org/html/rfc7292>

```
openssl pkcs12 -info -in ClientCertificate.pfx
```

It usually has the filename .p12 or .pfx. PFX and PKCS12 are sometimes used interchangeably.

## PEM

Pem is a simpler format than PKCS 12. It usually only includes a certificate and private key as base64 strings.

## Checks

### Access without certificate

Applications can be configured to ask for a client certificate without actually validating them. For example in F5 it is configured to "request" and not "required". <https://devcentral.f5.com/s/articles/How-Client-SSL-Authentication-works-on-BIG-IP>

# Send self-signed certificate

---

If the server looks at issuer or subject lines it might be possible to bypass it by simply copying those lines.

## Step 1: Create CA

```
openssl req -newkey rsa:4096 -keyform PEM -keyout ca.key -x509 -days 3650 -outform PEM -out ca.cer
```

## Step 2: Generate Client certificate

### Generate a key


```
openssl genrsa -out client.key 4096
```

### Generate a certificate signing request

```
openssl req -new -key client.key -out client.req
```

### Sign the CSR:

```
openssl x509 -req -in client.req -CA ca.cer -CAkey ca.key -set_serial 101 -extensions client -days 365 -outform PEM -out client.cer
```



## Step 3: Convert to PKCS12

```
openssl pkcs12 -export -inkey client.key -in client.cer -out client.p12
```

Here is a script that generates client-certificates with your specified issuer, subject and serialnumber.

```
#!/bin/bash
```

```
function generate_certificate {

    PASSWORD="pass"
    ISSUER="/C=SE/O=Example/OU=ExampleOU/CN=ExamplePurpose"
    SERIALNUMBER="100000000"
    SUBJECT="/C=SE/ST=Sweden/L=Stockholm/O=ExampleO/CN=test.test.com"

    echo "[*] Generating CA"

    openssl req -newkey rsa:4096 -keyform PEM -keyout ca.key -x509 -days 3650 -outform PEM -out ca.cer -subj "$ISSUER" -passout pass

    echo "[*] Reading the result"

    openssl x509 -in ca.cer -text -noout

    echo "[*] Generating client key"

    openssl genrsa -out client.key 4096

    echo "[*] Generating CSR from configuration file"

    openssl req -new -key client.key -out client.req -subj "$SUBJECT"

    echo "[*] Signing the CSR"

    openssl x509 -req -in client.req -CA ca.cer -CAkey ca.key -set_serial $SERIALNUMBER -extensions client -days 365 -outform PEM -o

    echo "[*] Transforming into p12 format"

    openssl pkcs12 -passout pass:$PASSWORD -export -inkey client.key -in client.cer -out client.p12

    echo "[*] Reading the final p12 certificate"

    openssl pkcs12 -passin pass:$PASSWORD -passout pass:$PASSWORD -info -in client.p12

    exit
}

generate_certificate
```

## Generate Client Certificate (s/mime) from a trusted CA

It is possible to create a S/MIME cert for free (or paid) associated with your email address. It might be that the backend only verifies that the client certificate is generated by a public CA. If that is the case it might work.

The default behaviour in IIS is to trust the windows default CA store. So if you generate a s/mime cert from comodo or whatever it might work.

You can generate a free s/mime certificate here:

<https://extrassl.actalis.it/portal/uapub/freemail?lang=en>